MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Memo 408

July 1977

THE FRL PRIMER

R. Bruce Roberts and Ira P. Goldstein

July 1977

The Frame Representation Language (FRL) is an experimental language written to explore the use of frames as a knowledge representation technique. The term "frame" as used in FRL was inspired by Minsky's [75] development of frame theory. FRL extends the traditional Property List representation scheme by allowing properties to have comments, defaults and constraints, to inherit information from abstract forms of the same type, and to have attached procedures triggered by adding or deleting values, or if a value is needed. We introduce FRL with the aid of a simple example: WHOSIS, a database of AI persons' names, addresses, interests and publications. A second section contains an abridged manual describing FRL's most-used commands and conventions.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. It was supported in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

Contents

Introduction

Figure 1 shows FRL from the larger perspective of intelligent support systems. FRL comprises the two bottom layers: a specialized data structure (the frame) and a collection of LISP functions for defining frames, storing and retrieving information. It has been used to implement NUDGE [Goldstein & Roberts 1977], a system for maintaining a person's schedule of activities in the face of individual preferences, conflicting constraints, and changing plans; PAL, a natural language front end for NUDGE [Bullwinkle 1977]; a system to assist planning a birthday party [Clemenson 1977]; TRIPPER, a knowledge base for places and travel around the country [Jeffery 1977]; a representation for the discourse structure of news articles [Rosenberg 1977]; and COMEX [Stansfield 1977], a system for understanding discourse about the commodities market.

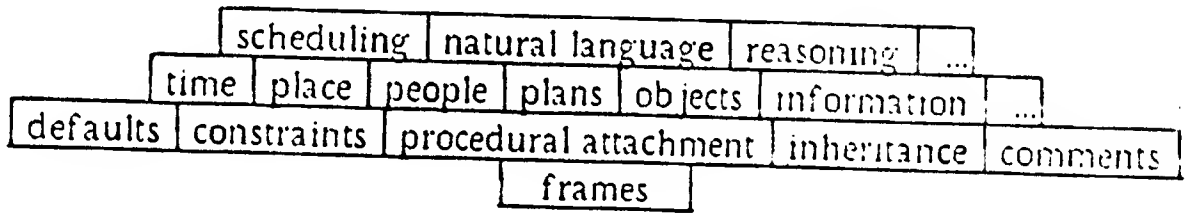


Figure 1 -- The Blocks World Model of FRL.

The intellectual issues surrounding the representation techniques provided by FRL are discussed in all of the papers cited above. The present document is intended only as a primer and consists of an introductory example of FRL's use and an abridged manual. An unabridged manual [Roberts & Goldstein 1977] is available.

Part I. The FRL Primer

This section illustrates FRL by describing WHOSIS, a frame system for answering questions about AI peoples' names, addresses, interests and publications. WHOSIS answers such questions as: What has Minsky published? Who at Stanford is interested in vision? What is Newell's zipcode? WHOSIS is, in essence, a generalized telephone directory organized in a hierarchical structure. The frames of WHOSIS are property lists, generalized by provision for inheritance, procedural attachment, constraints and defaults.

1. Define a frame with FASSERT.

The core of a FRL frame is a property list. For example, a WHOSIS individual is represented by such a structure, with properties for the name, address, interests and publications of that person. The frame for a representative AI person is created as follows:

```
(FASSERT MINSKY
    (NAME         ($VALUE ( |Marvin Minsky| )))
    (ADDRESS      ($VALUE ( |545 Technology Square, Room 821, Cambridge, MA 02139| )))
    (PUBLICATIONS ($VALUE ( |A Framework for Representing Knowledge.| )))
    (INTERESTS    ($VALUE ( REPRESENTATION ))))
```

FASSERT creates a frame named MINSKY having four slots, each with a single $VALUE facet. For example, |Marvin Minsky| is the value of the NAME slot. (In MACLISP, vertical bars are defined as a readmacro for creating non-standard atom names.)

2. Add information with FPUT.

To add another interest to MINSKY, annotating it with a comment specifying the source of the information, do:

```
(FPUT 'MINSKY 'INTERESTS '$VALUE 'ROBOTICS 'SOURCE: 'RBR)
```

Having done this, the frame for MINSKY looks like this:

```
(MINSKY
  ...
  (INTERESTS ($VALUE (REPRESENTATION)
                     (ROBOTICS (SOURCE: RBR))))
  ...)
```

The new value has been merged into the INTERESTS slot. The arguments to FPUT constitute an "indicator path", starting with the name of the frame and descending into the frame structure.

3. Information in a frame can have comments.

The Comment -- (SOURCE: RBR) -- consists of a "label" (conventionally ending in a ":") and a single "message". Messages, which lie at maximum depth in a frame structure, are not themselves embedded in a list.

4. Retrieve information from a frame with FGET.

Retrieving a value from a frame requires specifying the path which locates it in the frame structure. Thus,

(FGET 'MINSKY 'INTERESTS '$VALUE) returns (REPRESENTATION ROBOTICS), and

(FGET 'MINSKY 'INTERESTS '$VALUE 'ROBOTICS 'SOURCE:) returns (RBR).

Note that unless one explicitly maintains the inverse relations, finding the frames with a given slot value is much less efficient than finding the slot value of a known frame.

5. Delete information from a frame with FREMOVE.

To return the MINSKY frame to its original state, do:

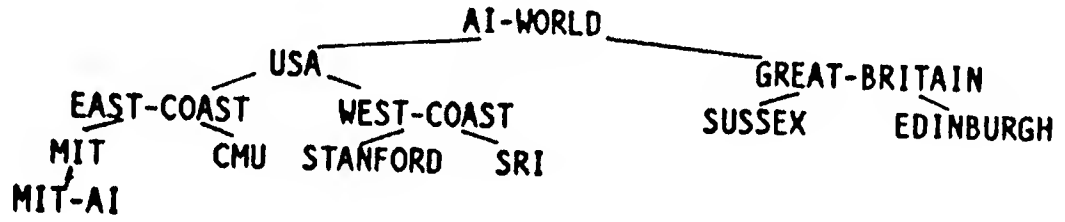
```
(FREMOVE 'MINSKY 'INTERESTS '$VALUE 'ROBOTICS)
```

FREMOVE accepts the same kind of indicator path as FPUT or FGET and excises the

portion of the frame lying at its terminus.

6. FRL provides inheritance.

FRL generalizes the property list representation by supplying a mechanism for inheritance. The advantage of inheritance is the efficient representation of shared properties. For example, many people will have approximately the same address, and we do not want this common information repeated for each individual. So, we break up the address into component parts, assigning each part to a different slot and allow people to inherit that portion of the address they share with others while supplying only the part unique to themselves. This is accomplished by organizing WHOSIS frames into a hierarchy of groups based on geographic proximity. Part of this hierarchy is shown below:



Each item in this inheritance hierarchy is a frame with its own attributes and values. For example, the generic frame for people at the MIT AI Laboratory is defined as follows:

```
(FASSERT MIT-AI
    (STREET ($VALUE ( |545 Technology Square| )))
    (CITY   ($VALUE ( |Cambridge| )))
    (STATE  ($VALUE ( |MA| )))
    (ZIP    ($VALUE ( |02139| ))))
```

Given this MIT-AI frame, the MINSKY frame can be redefined to assert only his office, with the remaining information about his address inherited from MIT-AI. This is done by indicating that MINSKY is AKO MIT-AI person.

```
(FREMOVE 'MINSKY 'ADDRESS)      ;The old address slot is deleted.
```

```
(FASSERT MINSKY
    ; FASSERT merges new information into an existing frame.
    (AKO            ($VALUE ( MIT-AI )))
    (OFFICE         ($VALUE ( 821 ))))
```

The AKO slot indicates to FGET that the MINSKY frame inherits values from MIT-AI. Thus, (FGET 'MINSKY 'CITY '$VALUE) now returns (|Cambridge|). FGET looks first in the frame explicitly mentioned as the first indicator in the path (MINSKY); then, as no value is found, FGET is reapplied to the frames named in the AKO slot (MIT-AI); and so on until a frame is encountered with a value for the CITY slot. Here MIT-AI supplies this value.

The advantages of inheritance are (1) information common to a set is represented explicitly only once, (2) there is no additional overhead in accessing just a part of the information, (3) partial information can be meaningfully represented if that is all that is known. The disadvantages are an increase in the time to retrieve the information due to following the AKO links and the need to reformat the pieces into the whole.

The AKO link establishes a mapping by which properties of one frame may be distributed to related frames. Semantically, the mapping can represent a set/subset, part/whole or other relationship. That AKO originally meant "a kind of" should not cause the reader to think that only abstraction relationships can be so represented. Subtleties of inheriting properties from set to subset, whole to part, and class to member must be addressed in more complex applications but carry us beyond the purposes of this primer. The WHOSIS example was chosen to avoid these subtleties. They are discussed in the papers cited earlier.

7. Retrieved information can be computed.

Having structured the WHOSIS world as a hierarchy, we need ways to once again access the address as a whole. One method is to define a function (GET-ADDRESS frame) that knows how the components of an address are distributed among the slots in a WHOSIS frame and returns the complete address. GET-ADDRESS is globally defined. It acts on frames and is known by name.

A second method is to make the value itself a procedure. This is accomplished by prefixing a value expression with a "!". Such expressions are evaluated when accessed by FGET. Thus we can define the address slot for all frames in WHOSIS by amending the AI-WORLD frame as follows:

```
(AI-WORLD
   ...
   (ADDRESS ($VALUE ( !(GET-ADDRESS :FRAME) )))
   ...)
```

MINSKY stands to inherit this procedure by virtue of the path:

MINSKY -> MIT-AI -> MIT -> EAST-COAST -> USA -> AI-WORLD.

To retrieve the complete address for MINSKY, one still does:

```
(FGET 'MINSKY 'ADDRESS '$VALUE)
```

except that now the information retrieved is the output of the GET-ADDRESS procedure. By convention, at the time the form is evaluated, the variable ":FRAME" is bound to MINSKY, the frame argument to FGET.

8. Procedures can be attached to a frame.

Functions can be associated directly with frames. These functions are triggered by the manipulation of the data in the frame: adding, removing, requesting information. In this case, the name of the function need not be known, only the conditions under which it

will be activated. The disadvantage with this scheme lies in the overhead of performing functions which truly are global in extent.

FRL provides the capability for the addition of a value to trigger the execution of procedures stored under the $IF-ADDED facet of the slot. The information in the $IF-ADDED facet is a list of functional expressions to be EVALed whenever a new value is added to its slot. For example, this mechanism could maintain a master list of interests espoused by people at MIT:

```
(FPUT 'MIT 'INTERESTS '$IF-ADDED '(PUSH :VALUE *MIT-MASTER-LIST*)).
```

Henceforth, any new value added to the INTERESTS slot of a frame subordinate to MIT will cause that value to be put on the *MIT-MASTER-LIST*. By convention, FRL binds the variable ":VALUE" to the value triggering the $IF-ADDED method, thus allowing the attached procedure to refer to the newly added value.

A complete scheme for maintaining the *MIT-MASTER-LIST* requires deleting an item if it is removed from anyone's INTERESTS slot. To accomplish this, FRL allows any slot to have an $IF-REMOVED facet, whose procedures are triggered whenever FREMOVE deletes a value from the slot. We supply such a procedure for the *MIT-MASTER-LIST* as follows:

```
(FPUT 'MIT 'INTERESTS '$IF-REMOVED '(POP :VALUE *MIT-MASTER-LIST*)).
```

$IF-ADDED and $IF-REMOVED facets inherit procedures via the AKO link. Consequently, the range of applicability of this knowledge can be controlled by its position in the inheritance hierarchy. It this inheritance that causes the *MIT-MASTER-LIST* maintenance programs to be triggered by the addition of an interest to any instance of the MIT frame.

Another form of procedural attachment is the $IF-NEEDED facet of a slot. A

common use of an attached $IF-NEEDED procedure is to manage the general task of instantiating a frame. Instantiation in FRL refers conceptually to generating an instance of a generic frame, a frame defining a entire class of objects. This entails creating the new instance frame within the frame system and assigning values to each of its slots. For example, MIT-AI is in fact a specific instance of a more general AI-WORLD frame.

```
(AI-WORLD
    (INSTANCE         ($IF-NEEDED ( (ADD-A-MEMBER) ))
                      ($VALUE     ( USA ) ( GREAT-BRITAIN ) ... ))
    (CLASSIFICATION ($IF-NEEDED ( (CLASSIFY) ))
                      ($VALUE     ( GENERIC )))
    (STREET           ($IF-NEEDED ( (ASK-FOR-STREET :FRAME) )))
    (CITY             ... )
    ... )
```

An INSTANCE slot in the AI-WORLD frame has an $IF-NEEDED procedure to administer the instantiation process:

```
(DEFUN ADD-A-MEMBER NIL
    (FINSTANTIATE :FRAME (REQUEST-NAME-FOR-INSTANCE)) ; Create new frame
    (MAPC '(LAMBDA (SLOT) (FNEED :FRAME SLOT))        ; Fill each slot
          (SETMINUS (FSLOTS :FRAME) '(AKO INSTANCE CLASSIFICATION))))
```

Attached procedures are inherited along the AKO link. The procedure defined in AI-WORLD can be used to create a new group within EAST-COAST, for example.

```
(FNEED 'EAST-COAST 'INSTANCE)
```

produces the following dialog:

What is the name of this new instance of EAST-COAST?
; FINSTANTIATE *asks for the name of the new frame?*
>BBN
Is BBN a new group or a person?
; CLASSIFY *asks for the classification of the new instance?*
>GROUP
Tell me the street of BBN?
; Now ADD-A-MEMBER *proceeds to run the $IF-NEEDED procedure*
; associated with the STREET *slot* — ASK-FOR-STREET.
; It then adds it as a value.
>50 Moulton Street

Tell me the city of BBN?
; *And so on for the remaining unfilled slots of* BBN.
...

The resulting newly created frame looks like this:

```
(BBN
    (AKO            ($VALUE ( EAST-COAST )))
    (CLASSIFICATION ($VALUE ( GENERIC )))       ; I.e., a group of people
    (STREET         ($VALUE ( |50 Moulton Street| )))
    (CITY           ($VALUE ( |Cambridge| )))
    ... )
```

The CLASSIFICATION slot distinguishes those frames containing information about a generic class of objects from those defining a single individual. It can have one of two values: INDIVIDUAL or GENERIC. Some instances of a frame that are simply further specializations while others represent actual members of the class. BBN is a generic frame. MINSKY is an individual.

The use of the "!" modifier discussed in the previous section to compute a value for FGET essentially defines a special $IF-NEEDED procedure, one encountered automatically by FGET whenever it looks up a value. FGET does not automatically call FNEED to evaluate all available methods.

9. Slots can have default values.

Frames can supply default data for a given attribute. This is represented by means of the $DEFAULT facet. In WHOSIS, for example, we use this capability to supply a default address for members of the AI Laboratory. This default is the director's office.

```
(FPUT 'MIT-AI 'OFFICE '$DEFAULT '817)
```

Defaults affect FGET's behavior. If no information is found in the $VALUE facet, either of the frame or any frame lying along its AKO link, FGET retries the frame and its superordinates, but looking in the $DEFAULT facet instead. Thus, if we remove

Minsky's office, (FREMOVE 'MINSKY 'OFFICE), then (FGET 'MINSKY 'OFFICE '$VALUE) returns (817), the default for MIT-AI.

10. Values can be constrained.

One can have information about a slot that restricts allowable values but does not actually supply one. FRL employs the $REQUIRE facet of a slot to represent this information. For example, in WHOSIS, the value of the STATE slot is constrained to be among a list of the fifty states or an accepted abbreviation. This would look like this:

```
(AI-WORLD
    ...
    (STATE ($REQUIRE ( (OR (MEMBER :VALUE STATE-ABBREVIATIONS) )))
                       (MEMBER :VALUE STATES))))
    ...)
```

Information in the $REQUIRE facet has the form of predicates which are true of allowable values. The variable ":VALUE" again is used to refer to the hypothetical value when writing the constraints. The consistency of a slot can be checked at any time using the function (FCHECK frame slot). We can write a function FPUT-CORRECT-VALUE that only adds a value if it satisfies all the constraints on the slot as follows:

```
(DEFUN FPUT-CORRECT-VALUE (FRAME SLOT NEW-VALUE)
       (AND (FCHECK FRAME SLOT NEW-VALUE)
            ; FCHECK's third argument is checked instead of the current value.
            (FPUT FRAME SLOT '$VALUE NEW-VALUE)))
```

11. Retrieve frames from a frame database by searching.

FQUERY, a search-oriented query function, retrieves frames which match a template. A request for people interested in Natural Language who work in CAMBRIDGE can be formulated as:

```
(FQUERY '(?
            (CITY      ($VALUE   ( CAMBRIDGE )))
            (INTERESTS ($VALUE   ( ?INTEREST ))
                       ($REQUIRE ( (AKO? :VALUE 'NATURAL-LANGUAGE) )))))
```

FQUERY's argument is a frame pattern, similar in appearance to a frame except for the occurrence of pattern variables, denoted by a prefix "?". The AKO? predicate returns T only if a value for the INTERESTS slot of a matching frame lies along the AKO link to NATURAL-LANGUAGE. Values must match, and requirements must be satisfied for a frame to match the pattern. FQUERY returns a list of matching frames along with their corresponding pattern variable bindings.

Retrieval from a frame database does not always require searching. For example, the hierarchical organization of WHOSIS demands that frames for people inhabit the "fringe" of the hierarchy. Thus, a subset of AI people in EAST-COAST is expressed as: (FRINGE 'EAST-COAST 'INSTANCE), where FRINGE returns the terminals of the tree defined with a root at EAST-COAST fanning out along the INSTANCE link. The predicate INDIVIDUAL? checks the CLASSIFICATION slot and is true for frames marked INDIVIDUAL. One can easily select those who have among their interests "NATURAL-LANGUAGE", for example:

```
(SELECT each PERSON in (FRINGE EAST-COAST 'INSTANCE) such that
    (AND (INDIVIDUAL? PERSON)
         (MEMBER 'NATURAL-LANGUAGE (FGET PERSON 'INTERESTS '$VALUE)))
```

where the lower case words are inserted here for readability.

SELECT is a straightforward LISP function illustrating that FRL is embedded in LISP and is not intended to supplant all features of the host language. Rather it only provides the means for manipulating a particular data structure within the larger LISP environment.

12. Save frames in a file.

Following a session in which the frame database has been altered, the user can save its current state as follows:

```
(FDUMP *FRAMES* '|DSK:ME;FILE DUMP|)
```

FRAMES is a global variable containing a list of all frames in the system. Alternatively, the user can save a subset of the defined frames by supplying his own list to FDUMP.

FDUMP writes a file of frame definitions which can be read with the normal Lisp reader, thereby recreating the state of the frame database.

13. FRL is a system program.

FRL exists as a dumped Lisp job and can be invoked by FRL^K. It initially tries to read a file -- DSK:user;.FRL. (INIT) -- analogous to Lisp.

In FRL, executing (WHOSIS) loads the database of people discussed in this primer. Documentation for WHOSIS is then available by typing (HELP). Try it, you'll like it as a computer-based yellow pages for the AI world.

Part II. The (abridged) FRL Manual

1. What is a FRAME?

A FRL frame is implemented as nested association lists with at most five levels of embedding. The respective sub-structures of a Frame are named: Slot, Facet, Datum, Comment and Message. The overall structure of a frame follows:

```
(frame1
  (slot1 (facet1 (datum1 (label1 message1 message2 ... more Messages ... )
                         ... more Comments ...)
                 (datum2 (label1 message1 ...))
                   ... more Data ...)
         (facet2 (datum1 (label1 message1 message2 ...)))
           ... more Facets ...)
  (slot2 (facet1 (datum1 (label1 message1 ...) ...)...)...)
    ... more Slots ...)
```

We will refer to the first element in one of these sub-structures as the *indicator* (said to name the structure) and the remaining elements collectively as the *bucket* (in the case of a slot, the bucket is a list of facets, for example). A path of indicators identifies a sub-structure in a frame. The order of sub-structures at any level in a frame is insignificant. In practice, facet names conventionally have a prefix "$"; labels, a suffix ":". This is simply to facilitate their recognition by the programmer.

2. Adding and Removing Frames.

(FASSERT *name slot1 slot2 ... slotN*)

creates a frame *name* (if it doesn't already exist) containing the slots *slot1 ... slotN*. If the *name* frame exists, the new information in the slots is merged with the existing slots. The frame is stored as the FRAME property of *name* and *name* is added to *FRAMES*, the list of known frames. FASSERT is a FEXPR.

The FASSERT switch. If FASSERT is nil, FASSERT forms are not interpreted. This is convenient for selectively reading just the code in a file containing intermixed code and frame definitions.

(FERASE *frame*)

removes *frame* from the FRAME property of its name and its name from the list *FRAME*.

Unless stated otherwise, a *frame* argument to a function can be either the name of a frame or the frame structure itself.

3. Frames and their names.

(FRAME *frame*)

returns a frame structure. An error if *frame* is neither a frame name nor a frame structure.

(FNAME *frame*)

returns the name of *frame*. An error if *frame* is neither a frame name nor a frame structure.

(FSLOTS *frame*)

returns a list of the slot names in *frame*.

4. The AKO and INSTANCE slots.

A slot is a generalization of the attribute-value pair in the traditional Property List representation. $VALUE is the slot facet which indicates its values. Five other "facets" indicate other types of knowledge associated with the slot. Data in the $DEFAULT facet supplies defaults. Data in $IF-ADDED and $IF-REMOVED facets are procedures triggered whenever a slot value is added or removed. $IF-NEEDED data are procedures which may compute a slot value. The $REQUIRE facet hold predicates which describe and restrict the value.

Two slots are recognized by FRL system functions: AKO (A Kind Of) and INSTANCE. These define a relation between frames along which data is inherited. The AKO relation is used to establish a conceptual hierarchy of frames in which general information stored higher in the hierarchy is inherited by more specialized concepts lower in the hierarchy. The INSTANCE slot is maintained as an inverse of the AKO pointer; thus `(FPUT 'A 'AKO 'B)` has the side-effect of asserting `(FPUT 'B 'INSTANCE 'A)`.

(FINSTANTIATE *frame {name}*)

creates an instance of *frame*; i.e., it possesses only an AKO link to *frame*. Its name is derived from the optional *name* argument and will be unique. The newly created frame is returned.

A relation between frames is defined by making the name of one frame the value of a slot in another frame. The slot names the relation. A tree of frame relations is possible since a slot can have many values. Several functions are provided to examine these relations.

(FCHILDREN *frame slot*)

returns a list of the immediate inferiors of *frame* along the relation named by *slot*.

This is just a list of values in *slot*.

(FTREE *frame slot*)

returns a tree of the form (root subtree1 subtree2 ...) with *frame* at the root; each subtree's root is a child of *frame* along the relation named by *slot*.

(FDESCENDANTS *frame slot*)

returns a list of all inferiors of *frame* along the relation *slot* defines. That is, it includes all the frames occurring in the "tree" of FTREE except the root *frame*.

(FRINGE *frame slot*)

returns a list of all "leaves" on the tree of (FTREE *frame slot*).

(FLINK? *relation f1 f2*)

Does *relation* connect *f1* to *f2*? *Relation* is a slot name whose values must be frame names; *f1* and *f2* are frames. (FLINK? 'AKO 'A 'B) is true only if a path exists from A to B following only the AKO "link"; i.e., if one of the values of the AKO slot of A is B, or FLINK? is true for any of these values.

(AKO? *f1 f2*)

returns T only if *f1* is a kind of *f2*. Equivalent to (FLINK? 'AKO *f1 f2*). Similar definitions are possible for any slot whose value is another frame.

5. Adding and Removing Parts of a Frame.

(FPUT *frame slot facet datum label message*)

adds the last argument at the point in *frame* named by the indicator path (the intervening arguments) and returns the modified *frame*. Adding new information to a frame is a merging process that retains the uniqueness of each indicator. FPUT is a LEXPR and can take from 2 to 6 arguments. It can be used to add an element anywhere in a frame; to add a *slot* name to *frame* or to put a *message* in a comment labeled *label*.

FPUT has an important side-effect. Putting data items into a $VALUE facet triggers the execution of all procedures in the $IF-ADDED facet of the slot.

(FREMOVE *frame slot facet datum label message*)

deletes the sub-structure of *frame* indicated by the path *slot -> facet -> datum* .... It returns the modified *frame*. FREMOVE is a LEXPR and can take from 2 to 6 arguments. The structure deleted will have had as its indicator the final argument to FREMOVE.

FREMOVE, too, has an important side-effect. If any data in a $VALUE facet is deleted by this command, all procedures in the $IF-REMOVED facet of the slot are executed.

6. The $IF-ADDED and $IF-REMOVED procedures.

Data in the $IF-ADDED and $IF-REMOVED facets is treated as evaluable LISP forms. The forms in the $IF-ADDED facet will be run whenever a value is added to the slot (i.e., in the $VALUE facet) by FASSERT or FPUT. The forms in the $IF-REMOVED facet will be run whenever a value is deleted from a slot (i.e., from the $VALUE facet) by FERASE or FREMOVE.

** All local and inherited procedures will be run.

** No $IF-ADDED procedure will be run if the value was already there. This serves to eliminate loops.

** No $IF-REMOVED procedure will be run if the value was not actually there to be removed.

** The procedures will be run in a Frame Environment in which the following global variables have been bound:

:FRAME = *frame*
:SLOT = *slot*
:FACET = $IF-ADDED or $IF-REMOVED (as appropriate).

In addition, the free variable ":VALUE" will be bound to the datum whose addition or removal precipitated the execution of the attached procedures.

7. Retrieving Parts of a Frame.

(FGET *frame slot facet datum label message*)

returns a list of all the indicators in the bucket addressed by the path of arguments. Usually, three arguments are given. The value of a slot is retrieved by (FGET *frame slot* '$VALUE). FGET looks first in the *slot* of *frame*. If data exists, a list of the items is returned. If no data is found, the facet of the frame named in *frame*'s AKO slot is inspected; and so on until a frame is found containing data, which is then returned.

An important special case is FGETting from a $VALUE facet. If still no value is found, FGET repeats, looking in the $DEFAULT facet instead.

** Inheritance stops at the first frame along the chain of AKO links that contains data.

** If FGET returns NIL, no data was found.

** A frame can be a-kind-of more than one other frame; i.e., have more than one value in its AKO slot. FGET traces each of the AKO paths, stopping at the first data encountered along each, and returns a list of all data thus found appended together.

** The FINHERIT Comment. A comment -- (FINHERIT: CONTINUE) -- on all datum structures in a facet causes the inheritance to proceed further along the AKO link as if no data had been found; it returns the local data appended to that found further along the link.

8. The $IF-NEEDED procedures.

All data in the $IF-NEEDED facets is treated as a LISP procedure. It can be run using the following function.

(FNEED *frame slot*)

executes the procedures in the $IF-NEEDED facet of *slot* in *frame*. The procedures will be run in a Frame Environment in which the following free variables have been bound:

:FRAME = *frame*, :SLOT = *slot*, :FACET = $IF-NEEDED

9. Constraining a Value.

Data items in the $REQUIRE facet should be a Lisp predicates which describe allowable values for the slot. There is an implicit conjunction between all data items present. The predicates are evaluated in the appropriate Frame Environment, like the other procedural knowledge already discussed. The variable :VALUES is bound to the list of values in question.

(FCHECK *frame slot {value}*)

returns a poll of all constraints in the $REQUIRE facet of *slot* in *frame* applied to the values of the slot. Both local and inherited constraints are included. If an optional *value* is supplied, it is checked against the constraints instead. Constraints are run in a Frame Environment with :FRAME, :SLOT and :VALUES bound. A poll is a list of four elements:

```
(<summary> <list of true predicates>
           <list of false predicates>
           <list of error-producing predicates>)
```

where the <summary> is T only if all are true, NIL only if some are false and none produce errors, and ? otherwise.

Two predicates test the classification of a frame. Frames in an AKO hierarchy are distinguished as being either GENERIC or INDIVIDUAL by the value of their CLASSIFICATION slot.

(INDIVIDUAL? *frame*)

returns T only if *frame* is marked as an individual. INDIVIDUAL? returns NIL if *frame* is generic, and ? otherwise.

(GENERIC? *frame*)

is defined analoguously to INDIVIDUAL?.

10. Saving and Restoring Frames.

(FDUMP *frames file*)

outputs to *file* each frame in the list *frames*. Frames so dumped can be read back into FRL to restore the state of the these frames in the data base. FDUMP creates a file of DEFRAME forms.

(DEFRAME *name slot1 slot2 ... slotN*)

defines a frame *name* containing precisely the slots *slot1 ... slotN*. Should *name* already exist, its previous definition is destroyed. The frame is stored as the FRAME property of *name* and *name* is added to *FRAMES*, the list of known frames. DEFRAME is a FEXPR.

The DEFRAME switch. If DEFRAME is nil, DEFRAME forms are not interpreted. This is convenient when for selectively reading a file with intermixed code and frame definitions.

Bibliography

Bullwinkle, C. "Levels of Complexity in Discourse," AI Memo 413, MIT, March 1977.

Clemenson, G. "A Birthday Party Frame System," AI Working Paper 140, MIT, February 1977.

Jeffery, M. "Representing PLACE in a Frame System," MS Thesis (forthcoming), MIT, 1977.

Goldstein, I.P. and Roberts, R.B. "NUDGE: A Knowledge-based Scheduling Program," AI Memo 405, MIT, February 1977.

Minsky, M. "A Framework for Representing Knowledge," in P. H. Winston (Ed.) *The Psychology of Computer Vision*, NY:McGraw-Hill, 1975.

Moon, D.A. *MACLISP Reference Manual.* LCS, MIT, December 1975.

Roberts, R.B. and Goldstein, I.P. "The FRL Manual," AI Memo 409, MIT, June 1977.

Rosenberg, S. "Frames-based Text Processing," AI Memo 431, MIT, 1977.

Stansfield, J. "COMEX: A Support System for a Commodities Expert," AI Memo 423, MIT, 1977.